Contents

Egress Gateways

13 minute read ✔ page test

This example does not work in Minikube.

The Accessing External Services task shows how to configure Istio to allow access to external HTTP and HTTPS services from applications inside the mesh. There, the external services are called directly from the client sidecar. This example also shows how to configure Istio to call external services, although this time indirectly via a dedicated *egress gateway* service.

Istio uses ingress and egress gateways to configure load balancers executing at the edge of a service mesh. An ingress gateway allows you to define entry points into the mesh that all incoming traffic flows through. Egress gateway is a symmetrical concept; it defines exit points from the mesh. Egress gateways allow you to apply Istio features, for example, monitoring and route rules, to traffic exiting the mesh.

Use case

Consider an organization that has a strict security requirement that all traffic leaving the service mesh must flow through a set of dedicated nodes. These nodes will run on dedicated machines, separated from the rest of the nodes running applications in the cluster. These special nodes will serve for policy enforcement on the egress traffic and will be monitored more thoroughly than other nodes.

Another use case is a cluster where the application nodes don’t have public IPs, so the in-mesh services that run on them cannot access the Internet. Defining an egress gateway, directing all the egress traffic through it, and allocating public

IPs to the egress gateway nodes allows the application nodes to access external services in a controlled way.

Before you begin

- Setup Istio by following the instructions in the Installation guide.

The egress gateway and access logging will be enabled if you install the `demo` configuration profile.

- Deploy the `sleep` sample app to use as a test source for sending requests. If you have `automatic sidecar injection` enabled, run the following command to deploy the sample app:

```
$ kubectl apply -f @samples/sleep/sleep.yaml@
```

Otherwise, manually inject the sidecar before deploying the `sleep` application with the following command:

```
$ kubectl apply -f <(istioctl kube-inject -f @samples/sleep/sleep.yaml
@)
```

You can use any pod with `curl` installed as a test source.

- Set the `SOURCE_POD` environment variable to the name of your source pod:

```
$ export SOURCE_POD=$(kubectl get pod -l app=sleep -o jsonpath={.items
..metadata.name})
```

- Enable Envoy's access logging

The instructions in this task create a destination rule for the egress gateway in the `default` namespace and assume that the client, `SOURCE_POD`, is also running in the `default` namespace. If not, the destination rule will not be found on the destination rule lookup path and the

client requests will fail.

Deploy Istio egress gateway

1. Check if the Istio egress gateway is deployed:

   ```
   $ kubectl get pod -l istio=egressgateway -n istio-system
   ```

   If no pods are returned, deploy the Istio egress gateway by performing the following step.

2. If you used an `IstioOperator` CR to install Istio, add the following fields to your configuration:

```
spec:
  components:
    egressGateways:
    - name: istio-egressgateway
      enabled: true
```

Otherwise, add the equivalent settings to your original `istioctl install` command, for example:

```
$ istioctl install <flags-you-used-to-install-Istio> \
                   --set components.egressGateways[0].name=istio-egres
sgateway \
                   --set components.egressGateways[0].enabled=true
```

Egress gateway for HTTP traffic

First create a `ServiceEntry` to allow direct traffic to an external service.

1. Define a `ServiceEntry` for `edition.cnn.com`.

`DNS` resolution must be used in the service entry below. If the resolution is `NONE`, the gateway will direct the traffic to itself in an infinite loop. This is because the gateway receives a request with the original destination IP address which is equal to the service IP of the gateway (since the request is directed by sidecar proxies to the gateway).

With `DNS` resolution, the gateway performs a DNS query to get an IP address of the external service

and directs the traffic to that IP address.

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: cnn
spec:
  hosts:
  - edition.cnn.com
  ports:
  - number: 80
    name: http-port
    protocol: HTTP
  - number: 443
    name: https
    protocol: HTTPS
  resolution: DNS
EOF
```

2. Verify that your `ServiceEntry` was applied correctly by sending an HTTP request to http://edition.cnn.com/politics.

```
$ kubectl exec "$SOURCE_POD" -c sleep -- curl -sSL -o /dev/null -D - h
ttp://edition.cnn.com/politics
...
HTTP/1.1 301 Moved Permanently
...
location: https://edition.cnn.com/politics
...

HTTP/2 200
Content-Type: text/html; charset=utf-8
...
```

   The output should be the same as in the TLS Origination for Egress Traffic example, without TLS origination.

3. Create an egress `Gateway` for *edition.cnn.com*, port 80, and

a destination rule for traffic directed to the egress gateway.

To direct multiple hosts through an egress gateway, you can include a list of hosts, or use `*` to match all, in the `Gateway`. The `subset` field in the `DestinationRule` should be reused for the additional hosts.

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: istio-egressgateway
spec:
```

```
  selector:
    istio: egressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - edition.cnn.com
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: egressgateway-for-cnn
spec:
  host: istio-egressgateway.istio-system.svc.cluster.local
  subsets:
  - name: cnn
EOF
```

4. Define a `VirtualService` to direct traffic from the sidecars to

the egress gateway and from the egress gateway to the external service:

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: direct-cnn-through-egress-gateway
spec:
  hosts:
  - edition.cnn.com
  gateways:
  - istio-egressgateway
  - mesh
  http:
  - match:
    - gateways:
      - mesh
      port: 80
    route:
    - destination:
```

```
        host: istio-egressgateway.istio-system.svc.cluster.local
        subset: cnn
        port:
          number: 80
      weight: 100
  - match:
    - gateways:
      - istio-egressgateway
      port: 80
    route:
    - destination:
        host: edition.cnn.com
        port:
          number: 80
      weight: 100
EOF
```

5. Resend the HTTP request to http://edition.cnn.com/politics.

```
$ kubectl exec "$SOURCE_POD" -c sleep -- curl -sSL -o /dev/null -D - h
ttp://edition.cnn.com/politics
...
HTTP/1.1 301 Moved Permanently
...
location: https://edition.cnn.com/politics
...

HTTP/2 200
Content-Type: text/html; charset=utf-8
...
```

The output should be the same as in the step 2.

6. Check the log of the `istio-egressgateway` pod for a line corresponding to our request. If Istio is deployed in the `istio-system` namespace, the command to print the log is:

```
$ kubectl logs -l istio=egressgateway -c istio-proxy -n istio-system |
 tail
```

You should see a line similar to the following:

```
[2019-09-03T20:57:49.103Z] "GET /politics HTTP/2" 301 - "-" "-" 0 0 90
 89 "10.244.2.10" "curl/7.64.0" "ea379962-9b5c-4431-ab66-f01994f5a5a5"
 "edition.cnn.com" "151.101.65.67:80" outbound|80||edition.cnn.com - 1
0.244.1.5:80 10.244.2.10:50482 edition.cnn.com -
```

Note that you only redirected the traffic from port 80 to the egress gateway. The HTTPS traffic to port 443 went directly to *edition.cnn.com*.

Cleanup HTTP gateway

Remove the previous definitions before proceeding to the next step:

```
$ kubectl delete gateway istio-egressgateway
$ kubectl delete serviceentry cnn
$ kubectl delete virtualservice direct-cnn-through-egress-gateway
$ kubectl delete destinationrule egressgateway-for-cnn
```

Egress gateway for HTTPS traffic

In this section you direct HTTPS traffic (TLS originated by the application) through an egress gateway. You need to specify port 443 with protocol `TLS` in a corresponding `ServiceEntry`, an egress `Gateway` and a `VirtualService`.

1. Define a `ServiceEntry` for `edition.cnn.com`:

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: cnn
spec:
  hosts:
  - edition.cnn.com
  ports:
  - number: 443
    name: tls
    protocol: TLS
  resolution: DNS
EOF
```

2. Verify that your `ServiceEntry` was applied correctly by sending an HTTPS request to https://edition.cnn.com/politics.

```
$ kubectl exec "$SOURCE_POD" -c sleep -- curl -sSL -o /dev/null -D - h
ttps://edition.cnn.com/politics
...
HTTP/2 200
Content-Type: text/html; charset=utf-8
...
```

3. Create an egress `Gateway` for *edition.cnn.com,* a destination rule and a virtual service to direct the traffic through the egress gateway and from the egress gateway to the external service.

To direct multiple hosts through an egress gateway, you can include a list of hosts, or use `*` to match all, in the `Gateway`. The `subset` field in the `DestinationRule` should be reused for the additional

hosts.

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: istio-egressgateway
spec:
  selector:
    istio: egressgateway
  servers:
  - port:
      number: 443
      name: tls
      protocol: TLS
    hosts:
    - edition.cnn.com
    tls:
      mode: PASSTHROUGH
---
```

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: egressgateway-for-cnn
spec:
  host: istio-egressgateway.istio-system.svc.cluster.local
  subsets:
  - name: cnn
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: direct-cnn-through-egress-gateway
spec:
  hosts:
  - edition.cnn.com
  gateways:
  - mesh
  - istio-egressgateway
  tls:
  - match:
    - gateways:
```

```
    - mesh
    port: 443
    sniHosts:
    - edition.cnn.com
  route:
  - destination:
      host: istio-egressgateway.istio-system.svc.cluster.local
      subset: cnn
      port:
        number: 443
- match:
  - gateways:
    - istio-egressgateway
    port: 443
    sniHosts:
    - edition.cnn.com
  route:
  - destination:
      host: edition.cnn.com
      port:
        number: 443
    weight: 100
```

```
EOF
```

4. Send an HTTPS request to https://edition.cnn.com/politics. The output should be the same as before.

```
$ kubectl exec "$SOURCE_POD" -c sleep -- curl -sSL -o /dev/null -D - h
ttps://edition.cnn.com/politics
...
HTTP/2 200
Content-Type: text/html; charset=utf-8
...
```

5. Check the log of the egress gateway's proxy. If Istio is deployed in the `istio-system` namespace, the command to print the log is:

```
$ kubectl logs -l istio=egressgateway -n istio-system
```

You should see a line similar to the following:

```
[2019-01-02T11:46:46.981Z] "- - -" 0 - 627 1879689 44 - "-" "-" "-" "-
" "151.101.129.67:443" outbound|443||edition.cnn.com 172.30.109.80:411
22 172.30.109.80:443 172.30.109.112:59970 edition.cnn.com
```

Cleanup HTTPS gateway

```
$ kubectl delete serviceentry cnn
$ kubectl delete gateway istio-egressgateway
$ kubectl delete virtualservice direct-cnn-through-egress-gateway
$ kubectl delete destinationrule egressgateway-for-cnn
```

Additional security considerations

Note that defining an egress `Gateway` in Istio does not in itself provides any special treatment for the nodes on which the egress gateway service runs. It is up to the cluster administrator or the cloud provider to deploy the egress gateways on dedicated nodes and to introduce additional security measures to make these nodes more secure than the rest of the mesh.

Istio *cannot securely enforce* that all egress traffic actually flows through the egress gateways. Istio only enables such flow through its sidecar proxies. If attackers bypass the

sidecar proxy, they could directly access external services without traversing the egress gateway. Thus, the attackers escape Istio's control and monitoring. The cluster administrator or the cloud provider must ensure that no traffic leaves the mesh bypassing the egress gateway. Mechanisms external to Istio must enforce this requirement. For example, the cluster administrator can configure a firewall to deny all traffic not coming from the egress gateway. The Kubernetes network policies can also forbid all the egress traffic not originating from the egress gateway (see the next section for an example). Additionally, the cluster administrator or the cloud provider can configure the network to ensure application nodes can only access the Internet via a gateway. To do this, the cluster administrator or the cloud provider can prevent the allocation of public IPs to pods other than gateways and can

configure NAT devices to drop packets not originating at the egress gateways.

Apply Kubernetes network policies

This section shows you how to create a Kubernetes network policy to prevent bypassing of the egress gateway. To test the network policy, you create a namespace, `test-egress`, deploy the sleep sample to it, and then attempt to send requests to a gateway-secured external service.

1. Follow the steps in the Egress gateway for HTTPS traffic section.
2. Create the `test-egress` namespace:

```
$ kubectl create namespace test-egress
```

3. Deploy the `sleep` sample to the `test-egress` namespace.

```
$ kubectl apply -n test-egress -f @samples/sleep/sleep.yaml@
```

4. Check that the deployed pod has a single container with no Istio sidecar attached:

```
$ kubectl get pod "$(kubectl get pod -n test-egress -l app=sleep -o js
onpath={.items..metadata.name})" -n test-egress
NAME                     READY     STATUS    RESTARTS   AGE
sleep-776b7bcdcd-z7mc4   1/1       Running   0          18m
```

5. Send an HTTPS request to https://edition.cnn.com/politics from the `sleep` pod in the `test-egress` namespace. The request will succeed since you did not define any restrictive policies yet.

```
$ kubectl exec "$(kubectl get pod -n test-egress -l app=sleep -o jsonp
ath={.items..metadata.name})" -n test-egress -c sleep -- curl -s -o /d
ev/null -w "%{http_code}\n"  https://edition.cnn.com/politics
200
```

6. Label the namespaces where the Istio components (the control plane and the gateways) run. If you deployed the Istio components to `istio-system`, the command is:

```
$ kubectl label namespace istio-system istio=system
```

7. Label the `kube-system` namespace.

```
$ kubectl label ns kube-system kube-system=true
```

8. Define a `NetworkPolicy` to limit the egress traffic from the `test-egress` namespace to traffic destined to `istio-system`, and to the `kube-system` DNS service (port 53):

```
$ cat <<EOF | kubectl apply -n test-egress -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-egress-to-istio-system-and-kube-dns
spec:
  podSelector: {}
  policyTypes:
  - Egress
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          kube-system: "true"
```

```
    ports:
    - protocol: UDP
      port: 53
  - to:
    - namespaceSelector:
        matchLabels:
          istio: system
EOF
```

Network policies are implemented by the network plugin in your Kubernetes cluster. Depending on your test cluster, the traffic may not be blocked in the following step.

9. Resend the previous HTTPS request to

https://edition.cnn.com/politics. Now it should fail since the traffic is blocked by the network policy. Note that the `sleep` pod cannot bypass `istio-egressgateway`. The only way it can access `edition.cnn.com` is by using an Istio sidecar proxy and by directing the traffic to `istio-egressgateway`. This setting demonstrates that even if some malicious pod manages to bypass its sidecar proxy, it will not be able to access external sites and will be blocked by the network policy.

```
$ kubectl exec "$(kubectl get pod -n test-egress -l app=sleep -o jsonp
ath={.items..metadata.name})" -n test-egress -c sleep -- curl -v -sS h
ttps://edition.cnn.com/politics
Hostname was NOT found in DNS cache
  Trying 151.101.65.67...
  Trying 2a04:4e42:200::323...
Immediate connect fail for 2a04:4e42:200::323: Cannot assign requested
 address
  Trying 2a04:4e42:400::323...
Immediate connect fail for 2a04:4e42:400::323: Cannot assign requested
 address
  Trying 2a04:4e42:600::323...
Immediate connect fail for 2a04:4e42:600::323: Cannot assign requested
 address
  Trying 2a04:4e42::323...
Immediate connect fail for 2a04:4e42::323: Cannot assign requested add
ress
connect to 151.101.65.67 port 443 failed: Connection timed out
```

10. Now inject an Istio sidecar proxy into the `sleep` pod in the `test-egress` namespace by first enabling automatic sidecar

proxy injection in the `test-egress` namespace:

```
$ kubectl label namespace test-egress istio-injection=enabled
```

11. Then redeploy the `sleep` deployment:

```
$ kubectl delete deployment sleep -n test-egress
$ kubectl apply -f @samples/sleep/sleep.yaml@ -n test-egress
```

12. Check that the deployed pod has two containers, including the Istio sidecar proxy (`istio-proxy`):

```
$ kubectl get pod "$(kubectl get pod -n test-egress -l app=sleep -o js
onpath={.items..metadata.name})" -n test-egress -o jsonpath='{.spec.co
ntainers[*].name}'
sleep istio-proxy
```

13. Create the same destination rule as for the `sleep` pod in the

`default` namespace to direct the traffic through the egress gateway:

```
$ kubectl apply -n test-egress -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: egressgateway-for-cnn
spec:
  host: istio-egressgateway.istio-system.svc.cluster.local
  subsets:
  - name: cnn
EOF
```

14. Send an HTTPS request to https://edition.cnn.com/politics. Now it should succeed since the traffic flows to `istio-egressgateway` in the `istio-system` namespace, which is allowed by the Network Policy you defined. `istio-`

`egressgateway` forwards the traffic to `edition.cnn.com`.

```
$ kubectl exec "$(kubectl get pod -n test-egress -l app=sleep -o jsonp
ath={.items..metadata.name})" -n test-egress -c sleep -- curl -sS -o /
dev/null -w "%{http_code}\n" https://edition.cnn.com/politics
200
```

15. Check the log of the egress gateway's proxy. If Istio is deployed in the `istio-system` namespace, the command to print the log is:

```
$ kubectl logs -l istio=egressgateway -n istio-system
```

You should see a line similar to the following:

```
[2020-03-06T18:12:33.101Z] "- - -" 0 - "-" "-" 906 1352475 35 - "-" "-
" "-" "-" "151.101.193.67:443" outbound|443||edition.cnn.com 172.30.22
3.53:39460 172.30.223.53:443 172.30.223.58:38138 edition.cnn.com -
```

Cleanup network policies

1. Delete the resources created in this section:

```
$ kubectl delete -f @samples/sleep/sleep.yaml@ -n test-egress
$ kubectl delete destinationrule egressgateway-for-cnn -n test-egress
$ kubectl delete networkpolicy allow-egress-to-istio-system-and-kube-d
ns -n test-egress
$ kubectl label namespace kube-system kube-system-
$ kubectl label namespace istio-system istio-
$ kubectl delete namespace test-egress
```

2. Follow the steps in the Cleanup HTTPS gateway section.

Troubleshooting

1. If mutual TLS Authentication is enabled, verify the correct certificate of the egress gateway:

```
$ kubectl exec -i -n istio-system "$(kubectl get pod -l istio=egressga
teway -n istio-system -o jsonpath='{.items[0].metadata.name}')"  -- ca
t /etc/certs/cert-chain.pem | openssl x509 -text -noout  | grep 'Subje
ct Alternative Name' -A 1
        X509v3 Subject Alternative Name:
            URI:spiffe://cluster.local/ns/istio-system/sa/istio-egress
gateway-service-account
```

2. For HTTPS traffic (TLS originated by the application), test the traffic flow by using the *openssl* command. *openssl* has an explicit option for setting the SNI, namely `-servername`.

```
$ kubectl exec "$SOURCE_POD" -c sleep -- openssl s_client -connect edi
tion.cnn.com:443 -servername edition.cnn.com
CONNECTED(00000003)
...
Certificate chain
 0 s:/C=US/ST=California/L=San Francisco/O=Fastly, Inc./CN=turner-tls.
map.fastly.net
   i:/C=BE/O=GlobalSign nv-sa/CN=GlobalSign CloudSSL CA - SHA256 - G3
 1 s:/C=BE/O=GlobalSign nv-sa/CN=GlobalSign CloudSSL CA - SHA256 - G3
   i:/C=BE/O=GlobalSign nv-sa/OU=Root CA/CN=GlobalSign Root CA
 ---
 Server certificate
 -----BEGIN CERTIFICATE-----
...
```

If you get the certificate as in the output above, your traffic is routed correctly. Check the statistics of the egress gateway's proxy and see a counter that corresponds to your requests (sent by *openssl* and *curl*) to

edition.cnn.com.

```
$ kubectl exec "$(kubectl get pod -l istio=egressgateway -n istio-syst
em -o jsonpath='{.items[0].metadata.name}')" -c istio-proxy -n istio-s
ystem -- pilot-agent request GET stats | grep edition.cnn.com.upstream
_cx_total
cluster.outbound|443||edition.cnn.com.upstream_cx_total: 2
```

Cleanup

Shutdown the sleep service:

```
$ kubectl delete -f @samples/sleep/sleep.yaml@
```